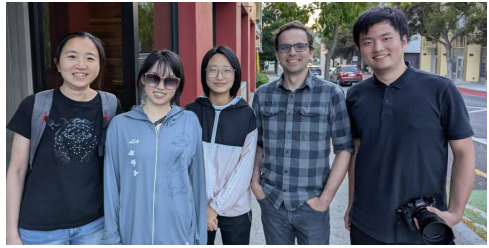
Updatable Private Set Intersection from Symmetric-Key Techniques

Junxin Liu	Oregon State University
Peihan Miao	Brown University
Mike Rosulek	Oregon State University
Xinyi Shi	Brown University
Jifeng Wang	Brown University



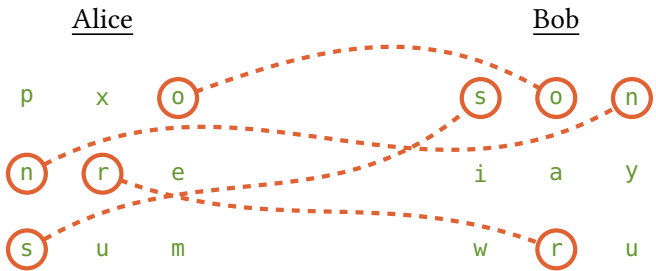
May 12: Eurocrypt 2026

what is private set intersection (PSI)?

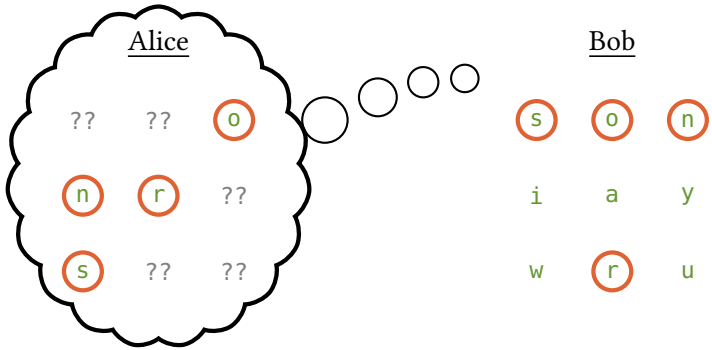
Alice		
p	x	o
n	r	e
s	u	m

Bob		
s	o	n
i	a	y
w	r	u

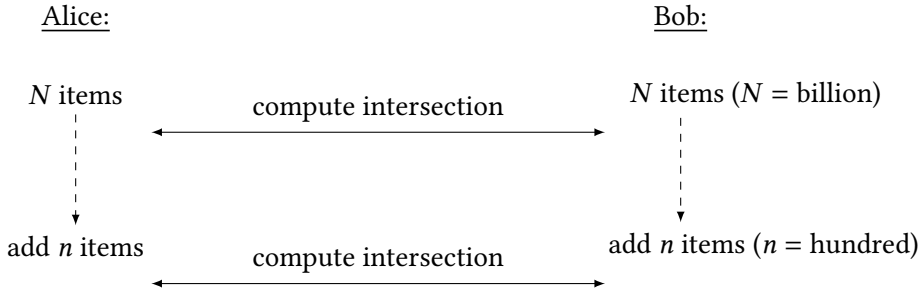
what is private set intersection (PSI)?

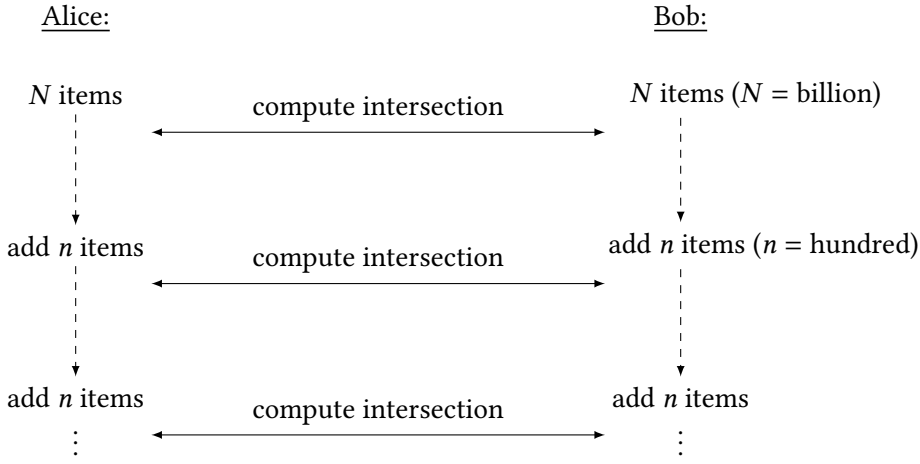


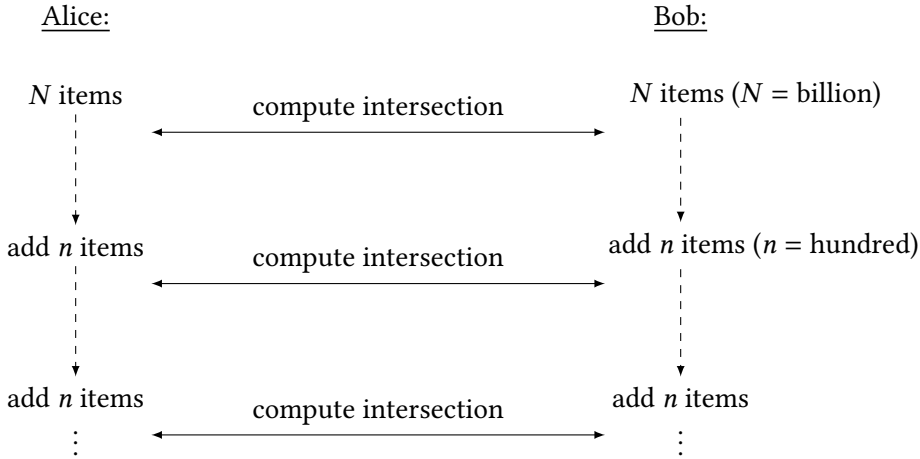
what is private set intersection (PSI)?




Alice:
Bob:
N items
compute intersection
N items (N = billion)


Alice:
Bob:
N items
compute intersection
N items (N = billion)
add n items
compute intersection
add n items (n = hundred)


Alice:
Bob:
N items
compute intersection
N items (N = billion)
add n items
compute intersection
add n items (n = hundred)
add n items
compute intersection
add n items



updatable PSI: *can marginal cost scale with n instead of N?*

[BadrinarayananMiaoXie22]

state of the art for updatable PSI:

- ✓ *sometimes* faster than recomputing plain PSI on N items
- × considerably more expensive than plain PSI on n items

[BadrinarayananMiaoXie22, BadrinarayananMiaoShiTromanhauserZeng24, AgarwalCashGeorgeKamaraMoatazSingh24, LingTangQiu24]

state of the art for updatable PSI:

- ✓ *sometimes* faster than recomputing plain PSI on N items
- × considerably more expensive than plain PSI on n items

[BadrinarayananMiaoXie22, BadrinarayananMiaoShiTromanhauserZeng24, AgarwalCashGeorgeKamaraMoatazSingh24, LingTangQiu24]

plain PSI protocols:

state-of-the-art protocols use exclusively **symmetric-key** techniques:

128 base OTs
+ $O(N)$ cheap symmetric-key operations

state of the art for updatable PSI:

- ✓ *sometimes* faster than recomputing plain PSI on N items
- × considerably more expensive than plain PSI on n items

[BadrinarayananMiaoXie22, BadrinarayananMiaoShiTromanhauserZeng24, AgarwalCashGeorgeKamaraMoatazSingh24, LingTangQiu24]

plain PSI protocols:

state-of-the-art protocols use exclusively **symmetric-key** techniques:

128 base OTs
+ $O(N)$ cheap symmetric-key operations

updatable PSI protocols:

all existing protocols rely on **public-key** techniques:

$O(n)$ expensive public-key operations

can we base updatable PSI
on symmetric-key techniques?

can we base updatable PSI on symmetric-key techniques?

yes!

all results in honest-but-curious model

can we base updatable PSI on symmetric-key techniques?

yes!

on sets with 4M items, 1K new items per epoch:

naïve (non-updatable) PSI [RaghuramanRindal22]	6 seconds
[BadrinarayananMiaoXie22]	3.2 seconds
[BadrinarayananMiaoShiTromanhauserZeng24]	99 seconds
our protocol	0.07 seconds

all results in honest-but-curious model

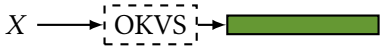
big picture: use state-of-the-art (plain) PSI as starting point

VOLE-PSI framework

[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16, PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]

VOLE-PSI framework

[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16,
PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]

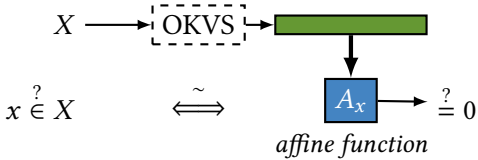


[linear] **OKVS**
= oblivious
key value store

encode a set into vector $\vec{v}$

VOLE-PSI framework

[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16,
PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]

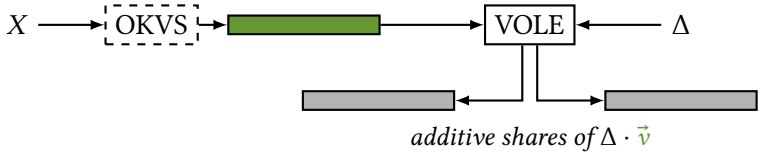


[linear] **OKVS**
= oblivious
key value store

encode a set into vector $\vec{v}$

VOLE-PSI framework

[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16,
PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]



[linear] **OKVS** = oblivious key value store

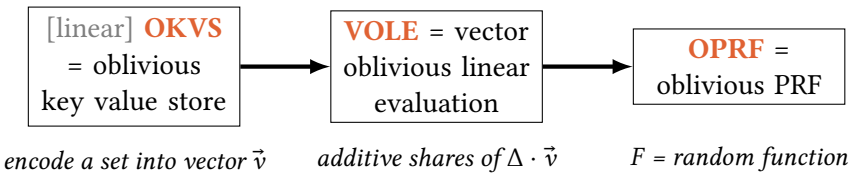
VOLE = vector oblivious linear evaluation

encode a set into vector $\vec{v}$

additive shares of $\Delta \cdot \vec{v}$

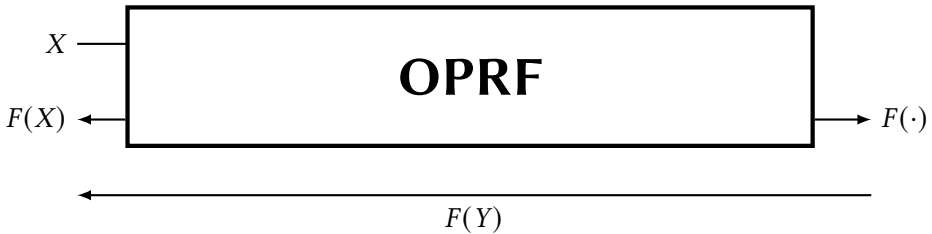
VOLE-PSI framework

[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16,
PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]



VOLE-PSI framework

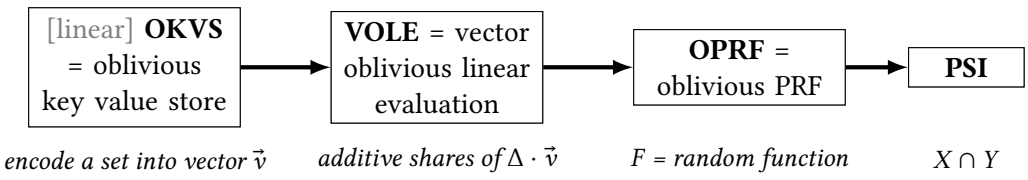
[PinkasSchneiderZohner14, KolesnikovKumaresanRosulekTrieu16, PinkasRosulekTrieuYanai20, GarimellaPinkasRosulekTrieuYanai21, RindalSchoppmann21]



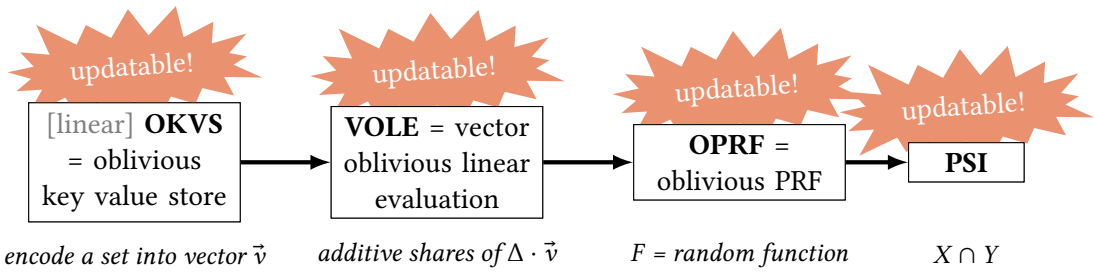
[linear] **OKVS** = oblivious key value store → **VOLE** = vector oblivious linear evaluation → **OPRF** = oblivious PRF → **PSI**

encode a set into vector $\vec{v}$ — *additive shares of* $\Delta \cdot \vec{v}$ — *F = random function* — $X \cap Y$

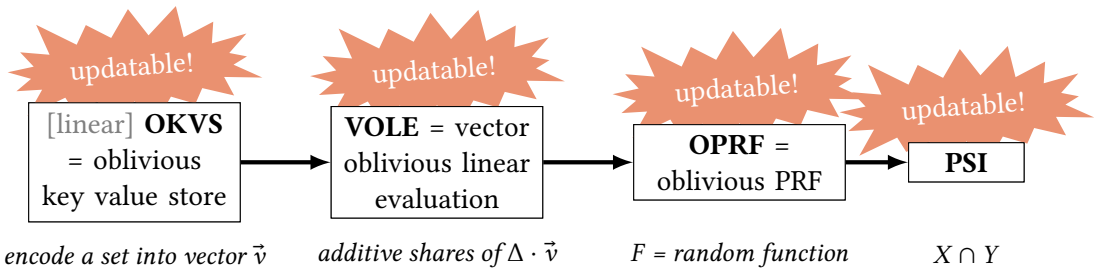
our approach



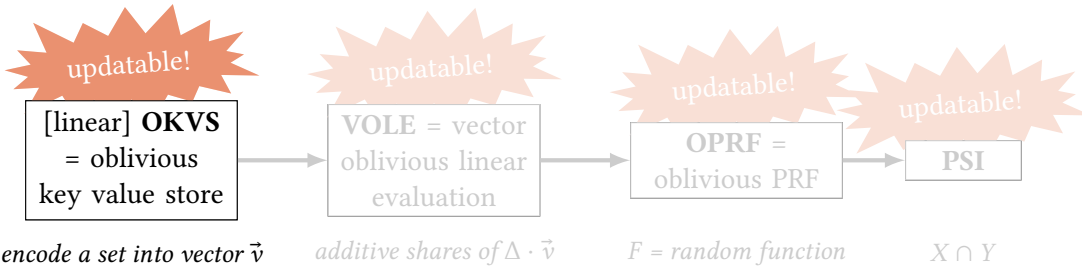
our approach



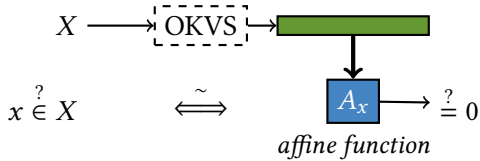
our approach



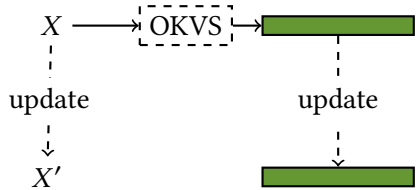
each step surprisingly non-trivial!


updatable!
[linear] **OKVS** = oblivious key value store
encode a set into vector $\vec{v}$
updatable!
VOLE = vector oblivious linear evaluation
additive shares of $\Delta \cdot \vec{v}$
updatable!
OPRF = oblivious PRF
F = random function
updatable!
PSI
$X \cap Y$

linear OKVS [GarimellaPinkasRosulekTrieuYanai21]

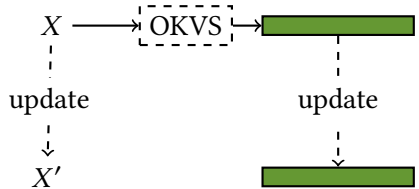


linear OKVS [GarimellaPinkasRosulekTrieuYanai21]



main idea: add/remove items from X by making a ***small number of changes*** *to the vector*

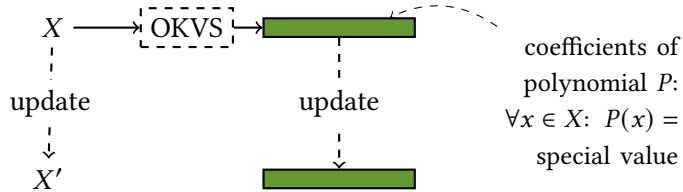
linear OKVS [GarimellaPinkasRosulekTrieuYanai21]



main idea: add/remove items from X by making a ***small number of changes*** *to the vector*

we couldn't figure out how to do this!

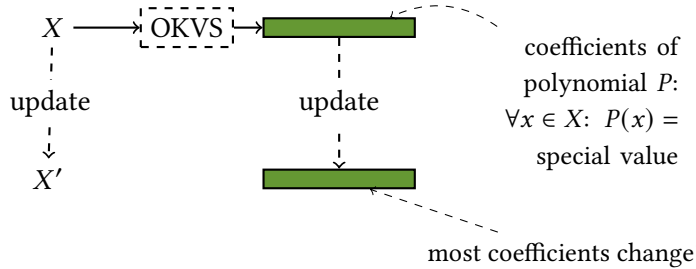
linear OKVS [GarimellaPinkasRosulekTrieuYanai21]



main idea: add/remove items from X by making a ***small number of changes*** *to the vector*

we couldn't figure out how to do this!

linear OKVS [GarimellaPinkasRosulekTrieuYanai21]

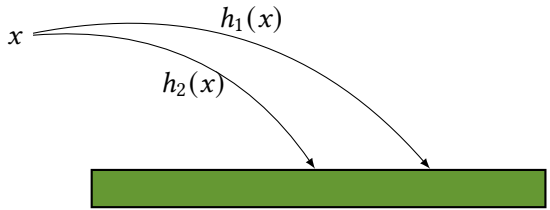


main idea: add/remove items from X by making a ***small number of changes*** *to the vector*

we couldn't figure out how to do this!

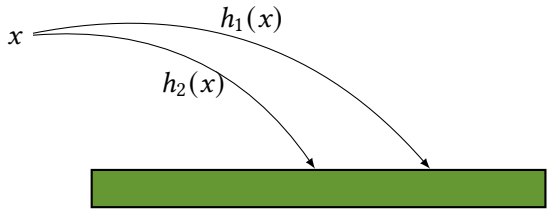
example: cuckoo hashing

example: cuckoo hashing



- **invariant:** $x \in X \iff$ "x" is at position $h_1(x)$ or $h_2(x)$

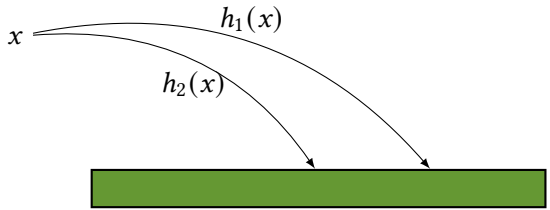
example: *cuckoo hashing*



- **invariant:** $x \in X \iff$ "x" is at position $h_1(x)$ or $h_2(x)$

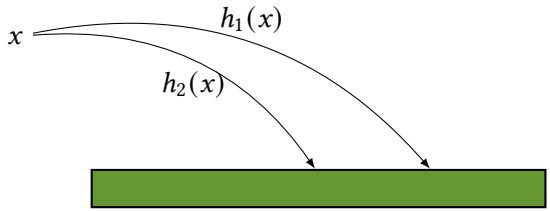
✓ add x by changing only few positions of vector (eviction)

example: cuckoo hashing

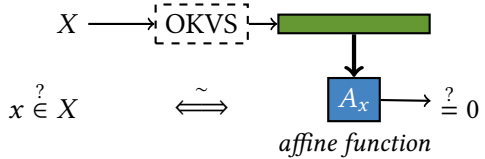


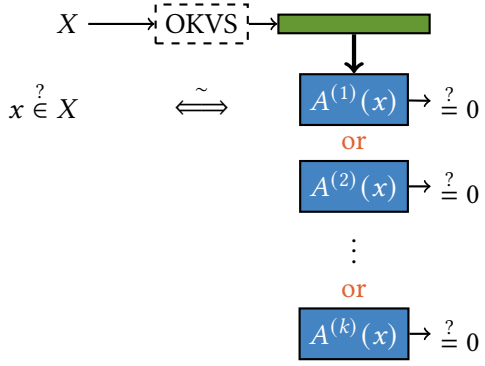
- **invariant:** $x \in X \iff$ "x" is at position $h_1(x)$ or $h_2(x)$
- ✓ add x by changing only few positions of vector (eviction)
- ✓ "does x exist at position i?" is affine

example: cuckoo hashing

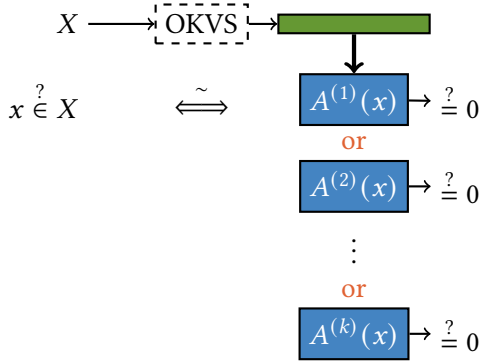


- ▶ **invariant:** $x \in X \iff$ "x" is at position $h_1(x)$ or $h_2(x)$
- ✓ add x by changing only few positions of vector (eviction)
- ✓ "does x exist at position i?" is affine
- × "is $x \in X$?" is **not affine!**


X
OKVS
$x \stackrel{?}{\in} X$
$\stackrel{\sim}{\Longleftrightarrow}$
A_x
$\stackrel{?}{=} 0$
affine function

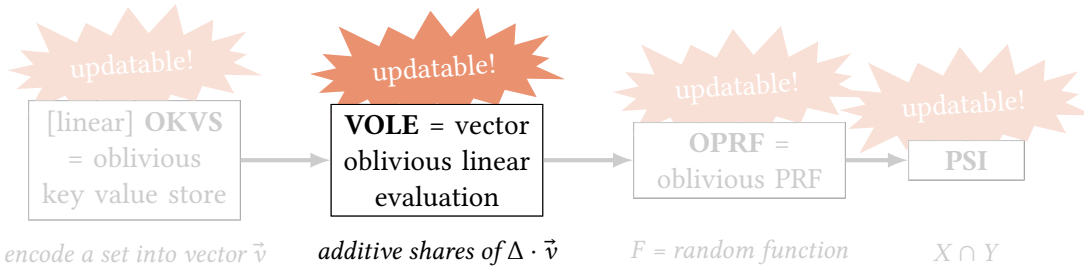


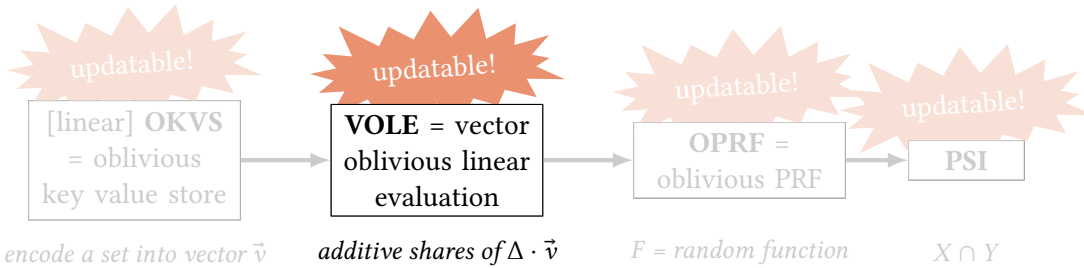
k-relaxed OKVS



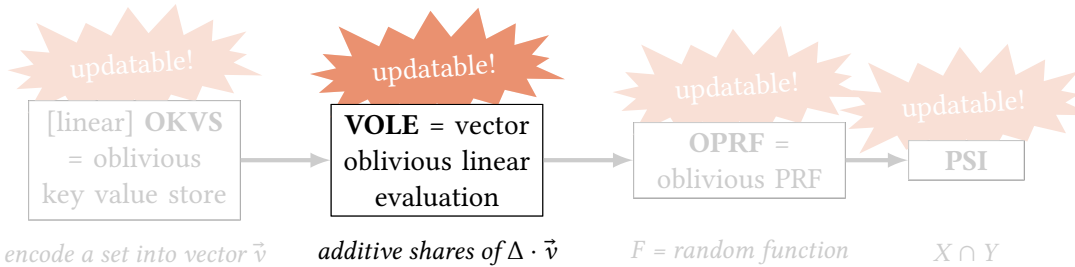
k-relaxed OKVS

efficient constructions with $k \in \{O(1), O(\log N), O(\log \log N)\}$


updatable!
[linear] OKVS = oblivious key value store
encode a set into vector $\vec{v}$
updatable!
VOLE = vector oblivious linear evaluation
additive shares of $\Delta \cdot \vec{v}$
updatable!
OPRF = oblivious PRF
F = random function
updatable!
PSI
$X \cap Y$

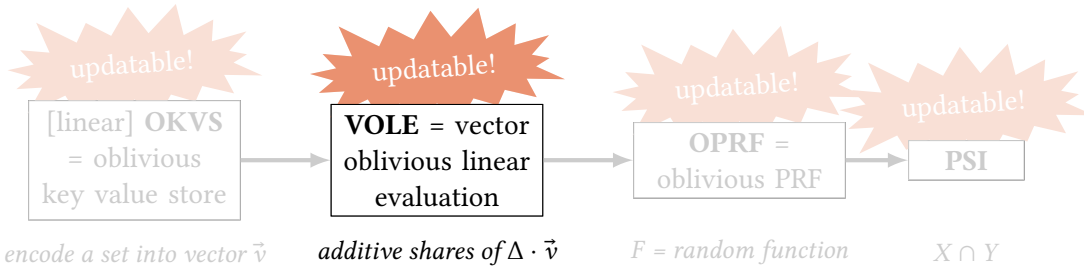


- updatable OKVS: encoding of $X \xrightarrow{\text{add items}}$ encoding of X'
- updatable VOLE: shares of $\Delta \cdot \vec{v} \xrightarrow{??\ ??\ ??}$ shares of $\Delta \cdot \vec{v}'$



- updatable OKVS: encoding of $X \xrightarrow{\text{add items}}$ encoding of X'
- updatable VOLE: shares of $\Delta \cdot \vec{v} \xrightarrow{??\ ??\ ??}$ shares of $\Delta \cdot \vec{v}'$

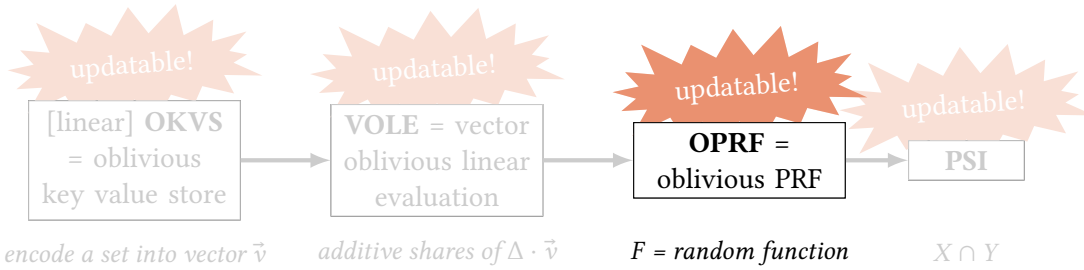
- $\vec{v}' - \vec{v}$ is a **sparse** vector!

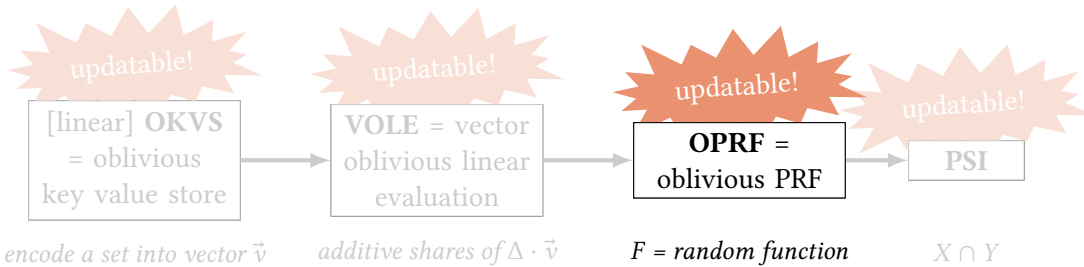


- updatable OKVS: encoding of $X \xrightarrow{\text{add items}}$ encoding of X'
- updatable VOLE: shares of $\Delta \cdot \vec{v} \xrightarrow{??\ ??\ ??}$ shares of $\Delta \cdot \vec{v}'$

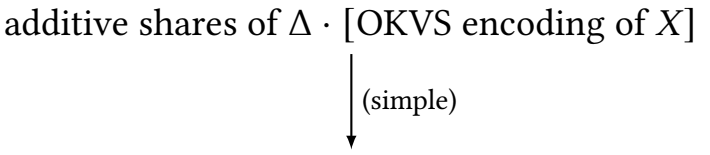
- $\vec{v}' - \vec{v}$ is a **sparse** vector!

$\Rightarrow$ additively share $\Delta \cdot (\vec{v}' - \vec{v})$ using [DoernerShelat17]
cost = $O(\lambda \log N)$


updatable!
[linear] OKVS = oblivious key value store
encode a set into vector $\vec{v}$
updatable!
VOLE = vector oblivious linear evaluation
additive shares of $\Delta \cdot \vec{v}$
updatable!
OPRF = oblivious PRF
F = random function
updatable!
PSI
$X \cap Y$



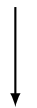
2 complications


additive shares of $\Delta \cdot$ [OKVS encoding of X]
(simple)
OPRF where receiver learns $F(X)$

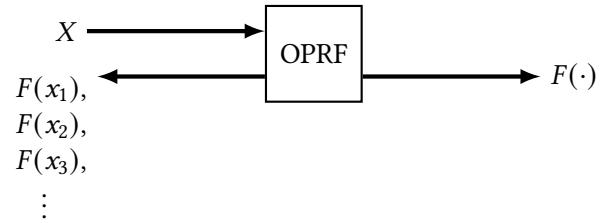
additive shares of $\Delta \cdot$ [k-relaxed OKVS encoding of X]

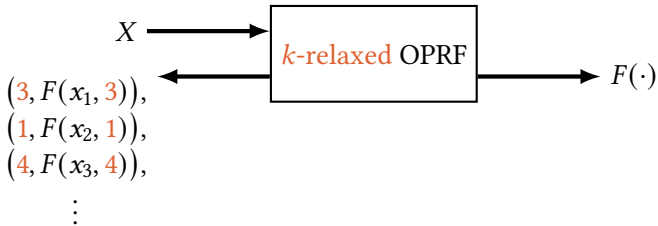
OPRF where receiver learns $F(X)$

additive shares of $\Delta \cdot$ [k-relaxed OKVS encoding of X]

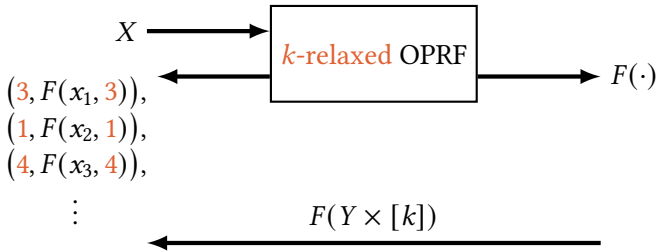


???


X
OPRF
F(x1),
F(x2),
F(x3),
F(·)



- for each x: associated outputs $F(x, 1), \ldots, F(x, k)$
- if $x \in X$: receiver learns *some* $F(x, i)$
- sender can compute *any* $F(x, i)$



- for each x: associated outputs $F(x, 1), \ldots, F(x, k)$
- if $x \in X$: receiver learns *some* $F(x, i)$
- sender can compute *any* $F(x, i)$

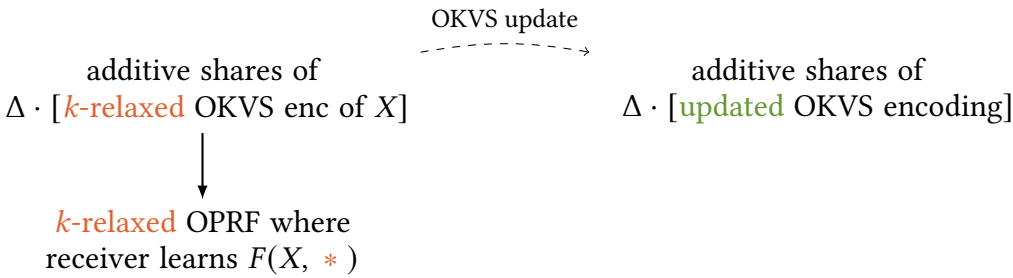
✓ still useful for PSI (but $k\times$ communication)

another problem!

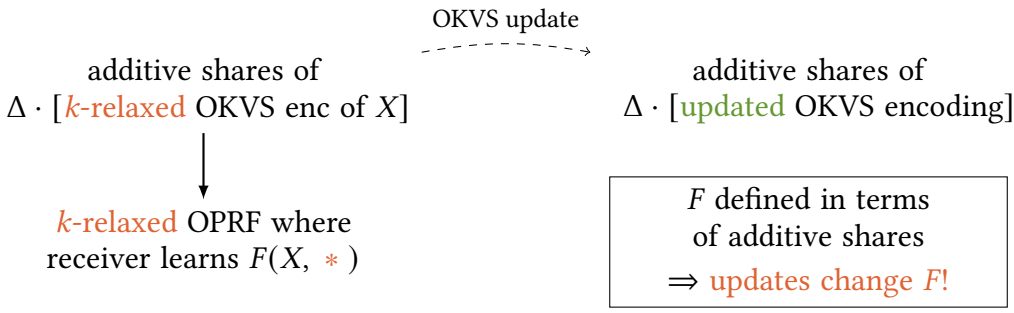
additive shares of
$\Delta \cdot$ [k-relaxed OKVS enc of X]

k-relaxed OPRF where
receiver learns $F(X, *)$

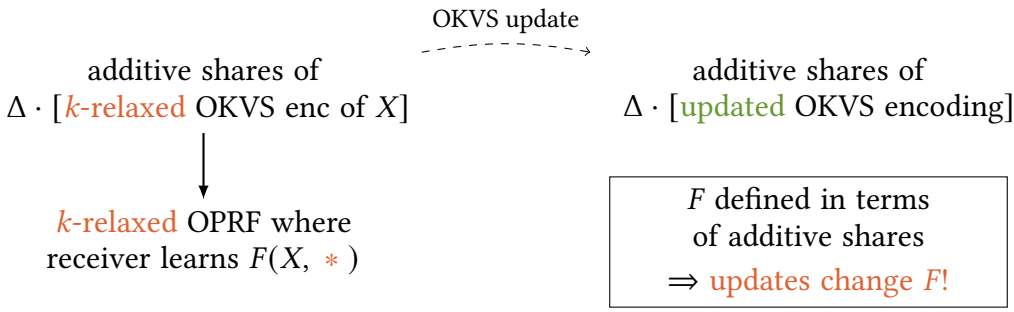
another problem!



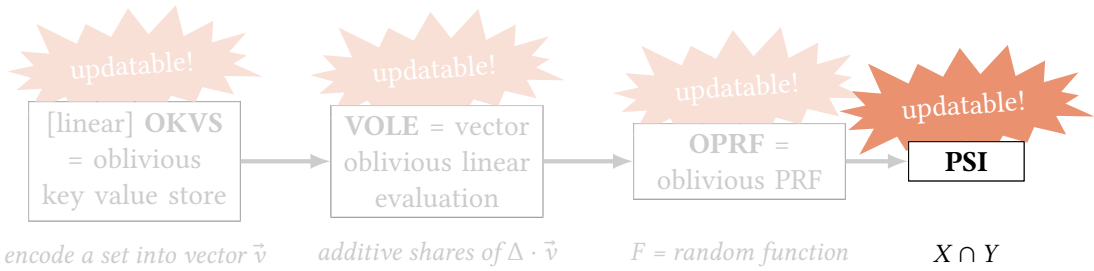
another problem!



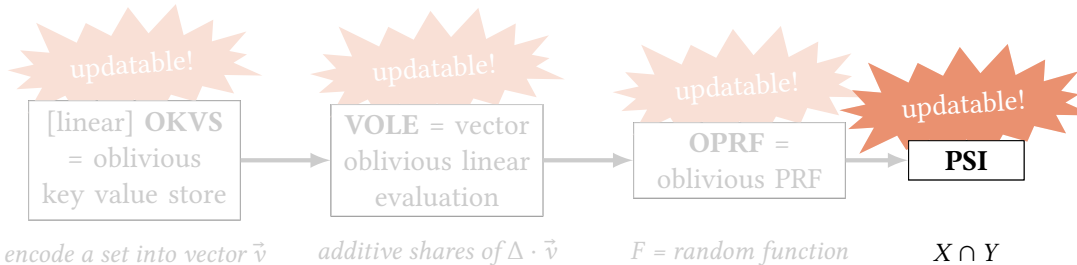
another problem!



- parties must learn which PRF outputs have changed!
- highly non-trivial to define *updatable* OPRF

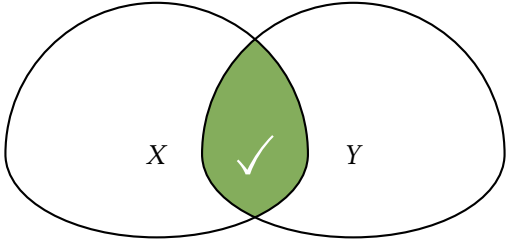


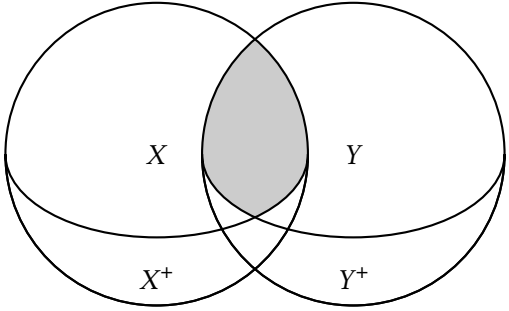
given an updatable OPRF, how to get updatable PSI?

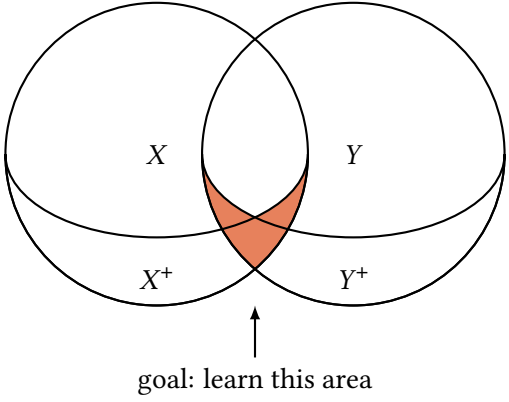


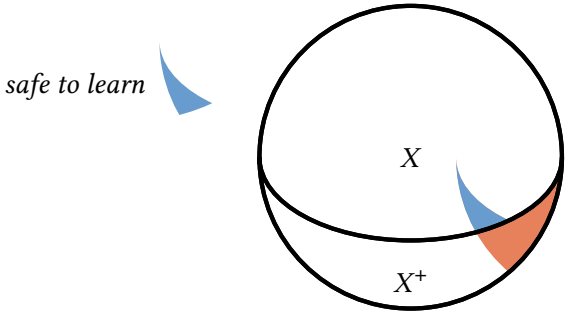
given an updatable OPRF, how to get updatable PSI?

(for now: focus on additions only)

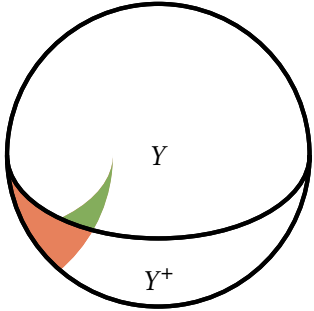

X
Y


X
Y
X+
Y+


X
Y
X+
Y+
goal: learn this area

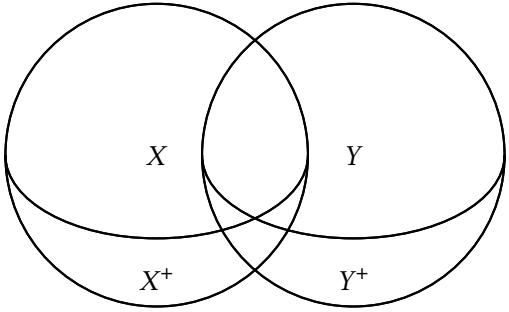

safe to learn
X
X^+

safe to learn

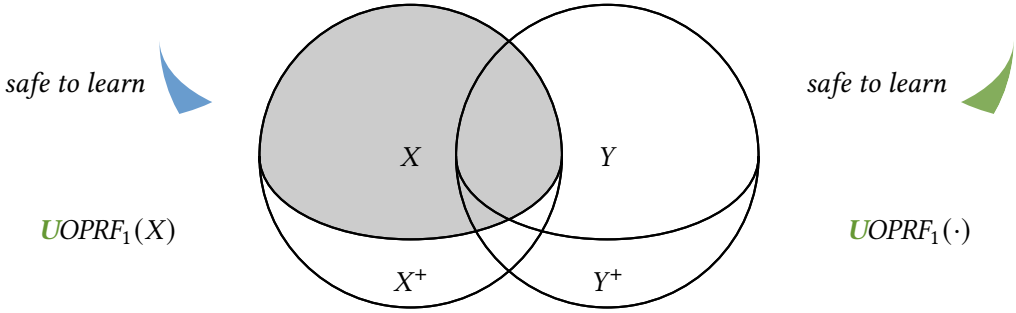


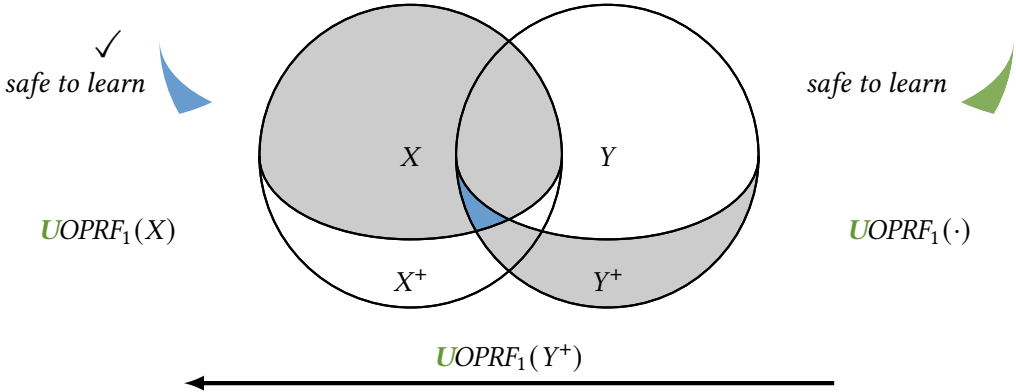
safe to learn

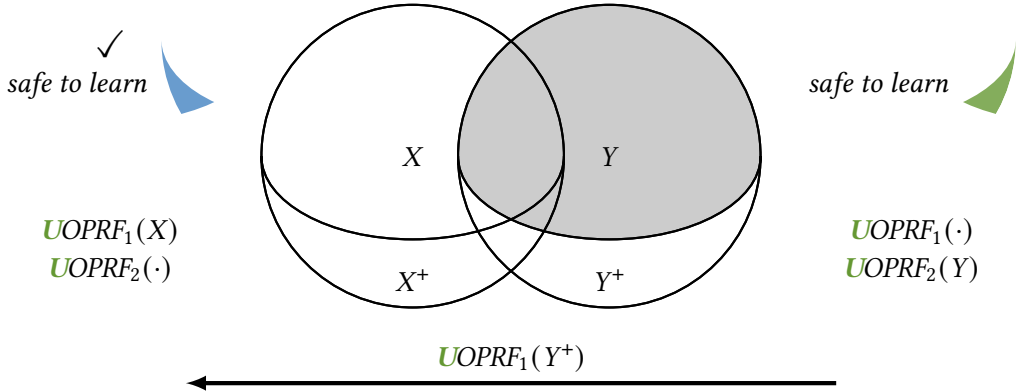
safe to learn

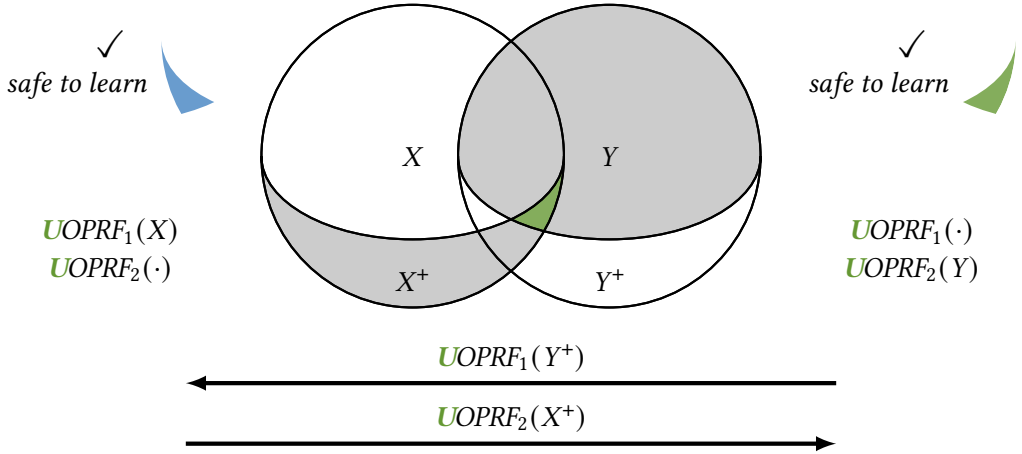


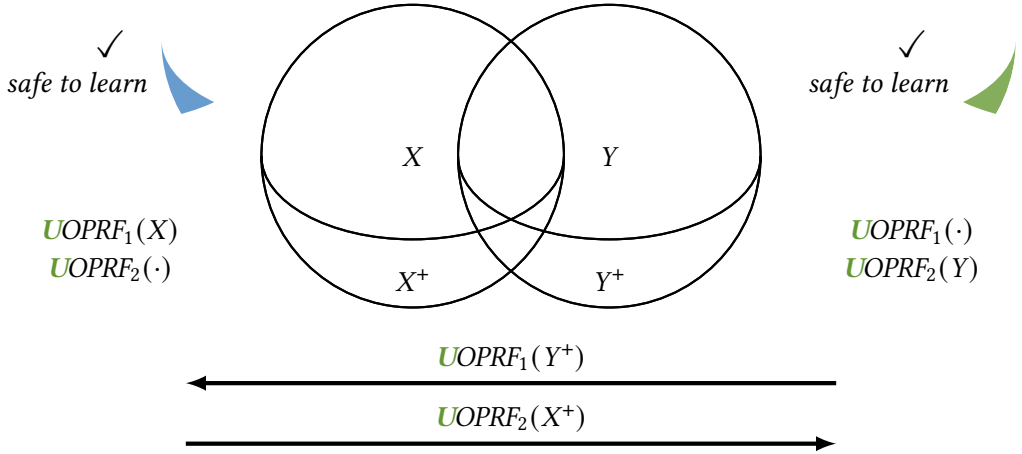
safe to learn

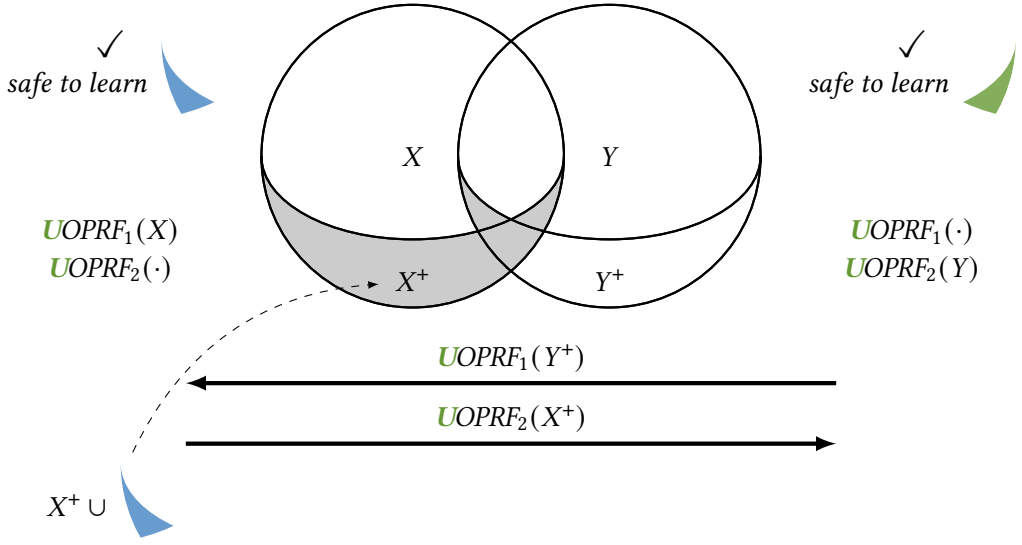

safe to learn
safe to learn
X
Y
UOPRF1(X)
UOPRF1(·)
X+
Y+

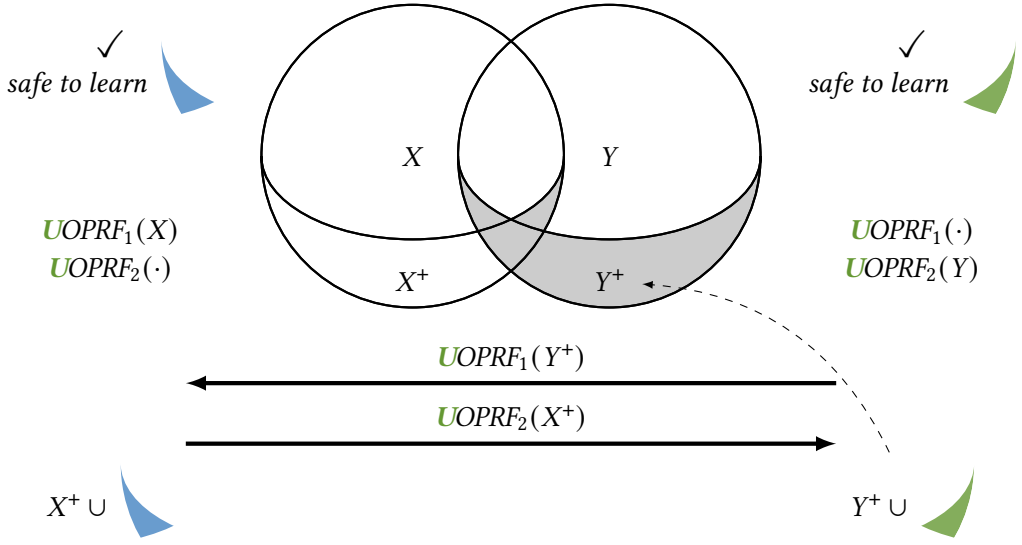

✓
safe to learn
safe to learn
X
Y
UOPRF1(X)
UOPRF1(·)
X+
Y+
UOPRF1(Y+)

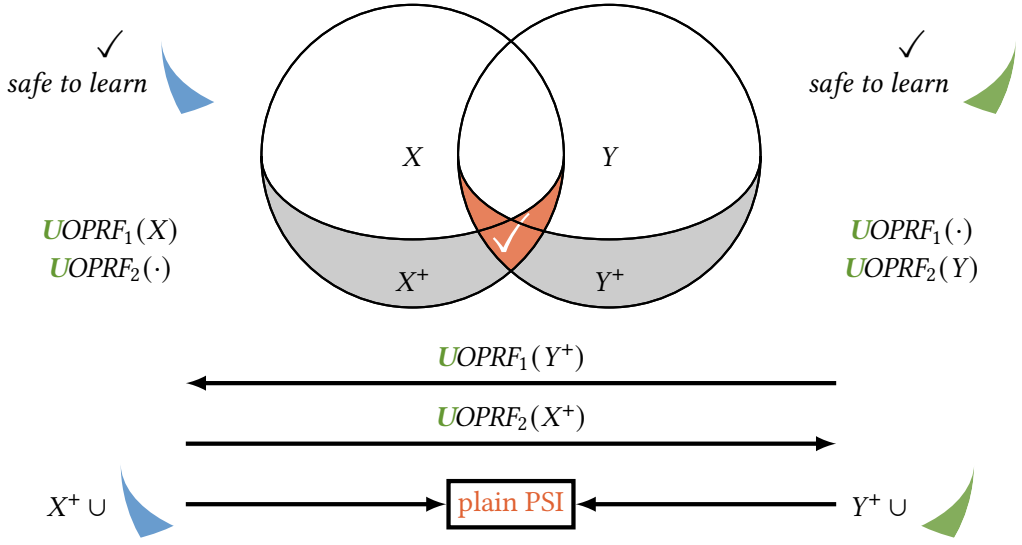

✓
safe to learn
safe to learn
X
Y
UOPRF1(X)
UOPRF2(·)
UOPRF1(·)
UOPRF2(Y)
X+
Y+
UOPRF1(Y+)

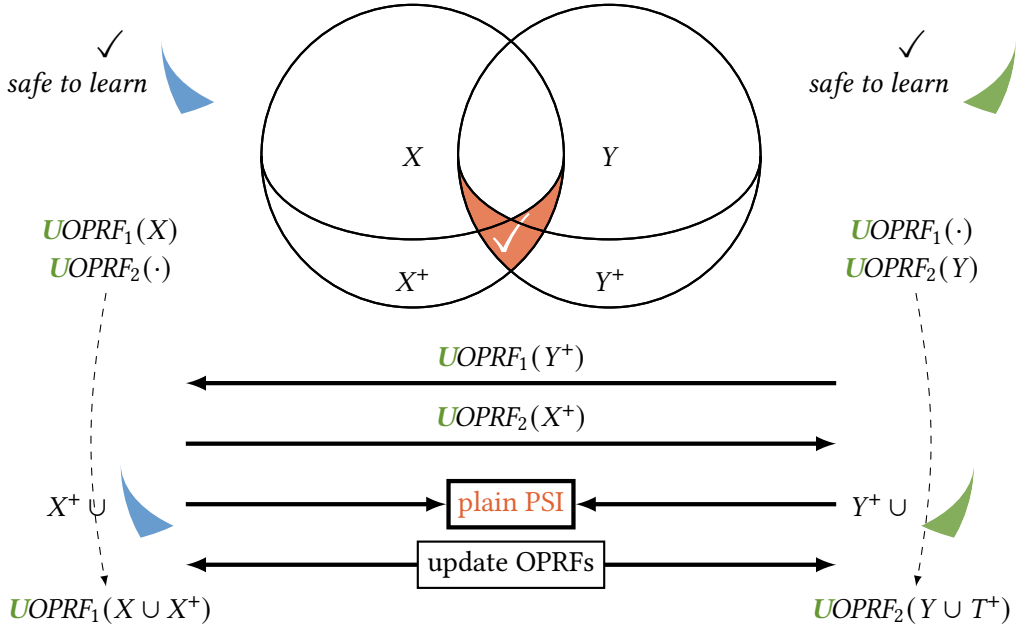

✓
safe to learn
X
Y
✓
safe to learn
UOPRF1(X)
UOPRF2(·)
X+
Y+
UOPRF1(·)
UOPRF2(Y)
UOPRF1(Y+)
UOPRF2(X+)


✓
safe to learn
X
Y
✓
safe to learn
UOPRF1(X)
UOPRF2(·)
X+
Y+
UOPRF1(·)
UOPRF2(Y)
UOPRF1(Y+)
UOPRF2(X+)


✓
safe to learn
✓
safe to learn
X
Y
UOPRF1(X)
UOPRF2(·)
UOPRF1(·)
UOPRF2(Y)
X+
Y+
UOPRF1(Y+)
UOPRF2(X+)
X+ ∪

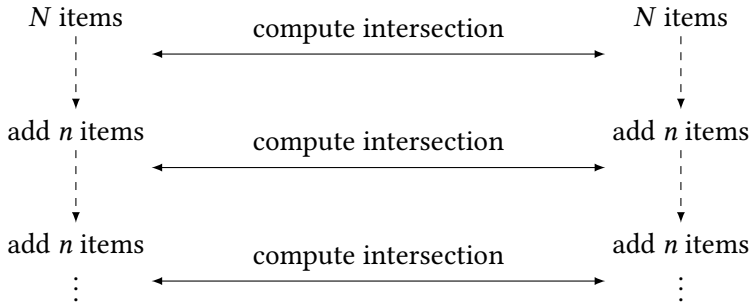

✓
safe to learn
✓
safe to learn
X
Y
UOPRF1(X)
UOPRF2(·)
UOPRF1(·)
UOPRF2(Y)
X+
Y+
UOPRF1(Y+)
UOPRF2(X+)
X+ ∪
Y+ ∪


✓
safe to learn
✓
safe to learn
X
Y
UOPRF1(X)
UOPRF2(·)
UOPRF1(·)
UOPRF2(Y)
X+
Y+
UOPRF1(Y+)
UOPRF2(X+)
X+ ∪
plain PSI
Y+ ∪

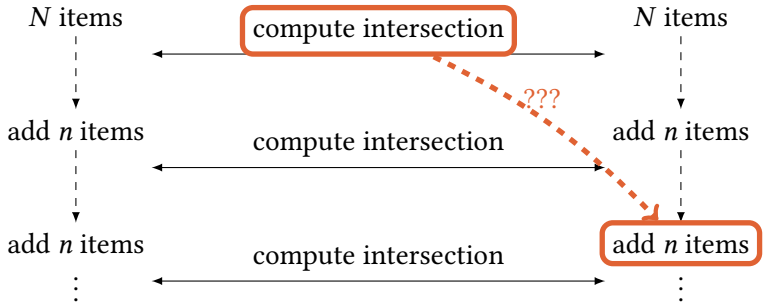

✓
safe to learn
X
Y
✓
safe to learn
X+
Y+
UOPRF1(X)
UOPRF2(·)
UOPRF1(·)
UOPRF2(Y)
UOPRF1(Y+)
UOPRF2(X+)
X+ ∪
plain PSI
Y+ ∪
update OPRFs
UOPRF1(X ∪ X+)
UOPRF2(Y ∪ T+)

a bonus result from our paper

a bonus result from our paper

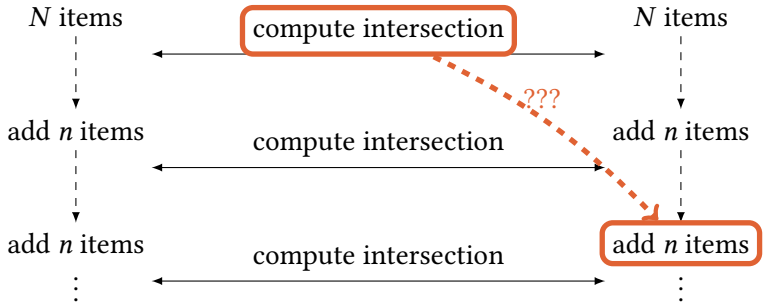


a bonus result from our paper



can choice of new items depend on previous protocol messages?

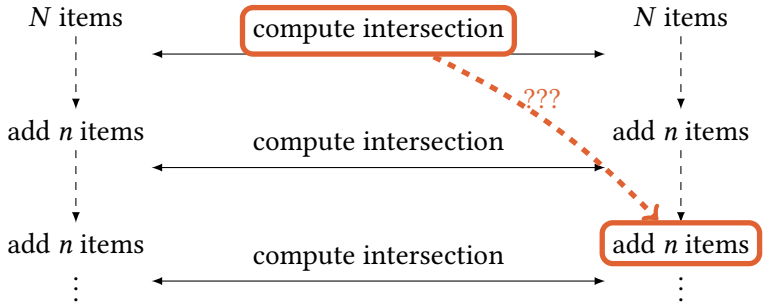
a bonus result from our paper



can choice of new items depend on previous protocol messages?

- **no**, in the security model of all prior works

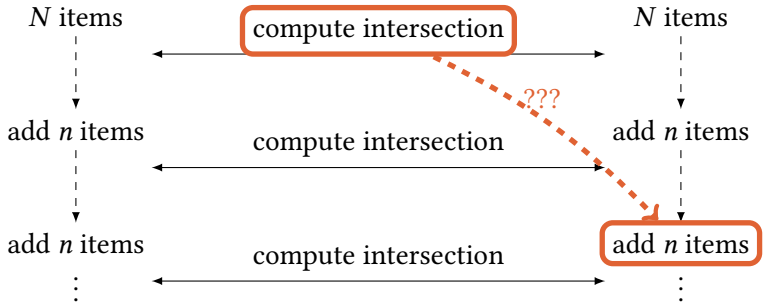
a bonus result from our paper



can choice of new items depend on previous protocol messages?

- ▶ **no**, in the security model of all prior works
- !! some protocols trivially broken by adaptive updates

a bonus result from our paper

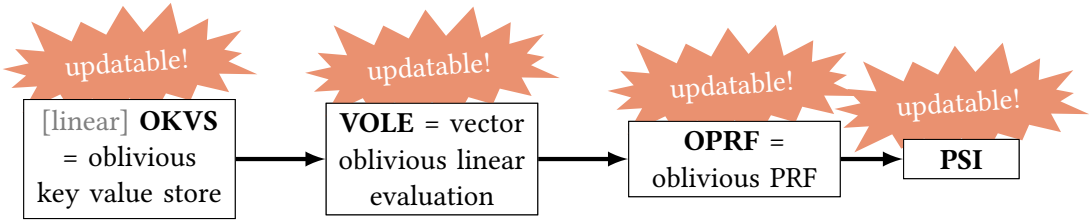


can choice of new items depend on previous protocol messages?

- ▶ **no**, in the security model of all prior works
- !! some protocols trivially broken by adaptive updates
- ✓ we explicitly consider adaptive updates, and show how to support

summary

first updatable PSI based on fast symmetric-key techniques

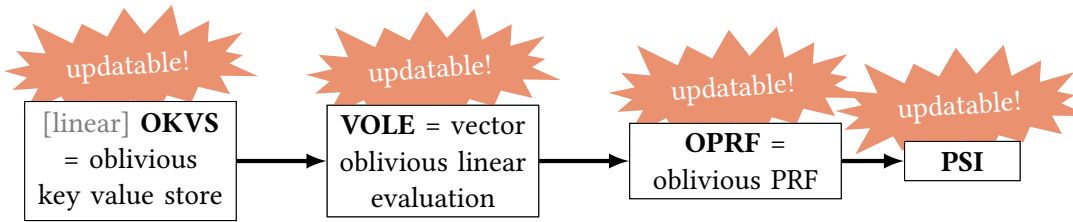


open problems:

- more efficient **deletion**: (we require $O(N)$ *computation* for deletions)
- security against malicious adversaries?

summary

first updatable PSI based on fast symmetric-key techniques



open problems:

- more efficient **deletion**: (we require $O(N)$ *computation* for deletions)
- security against malicious adversaries?

thanks for your attention!